A regime of droit moral detached from software copyright - the undeath of the ‘author’ in free and open source software licensing

Zhu, Chen

DOI:
10.1093/ijlit/eau004

Document Version
Peer reviewed version

Citation for published version (Harvard):
Zhu, C 2014, 'A regime of *droit moral* detached from software copyright - the undeath of the ‘author’ in free and open source software licensing', *International Journal of Law and Information Technology*, vol. 22, no. 4, pp. 367-392. https://doi.org/10.1093/ijlit/eau004

A Regime of *Droit Moral* Detached from Software Copyright? —The Undeath of the 'Author' in Free and Open Source Software Licensing

Chen Wei Zhu

School of Law

University of Birmingham

Published in International Journal of Law and Information Technology (2014) 22, 367–392
doi: 10.1093/ijlit/eau004

1. Introduction
2. A 'Private' Moral Right Regime?
2.1 From the Hacker Ethic to Open Source: A Very Brief History
2.2 Filling the Lacuna: The Prevalent Attribution Requirement
3. Rehabilitating Craftsmanship: Authorial Personas of FOSS Programmers
3.1 The Lingering Romantic Aesthetic in Software Copyright
3.2 The Postmodern Critique of the Romantic Author
3.3 Engaging with the Code: Craftsmanship and FOSS Programming
3.3.1 Two Traits of Programmer as Craftsman
3.3.2 Differences between Craftsmanship and Postmodernism
3.4 Intermediate Conclusion: Why Does Craftsmanship Matter?
4. Re-inventing the Legal Persona for FOSS Authorship
4.1 *Jacobsen v Katzer*: Failure to Depart from 'Authorship as Property'
4.2 Crafting Stewardship: A Normative Call for 'Authorship as Responsibility'
5. Conclusion

1. Introduction

Who are the 'authors' of free and open source software (FOSS) projects such as the iconic GNU/Linux operating system? How do FOSS programmers claim 'authorship' in their collaboratively created code through their licensing schemes (including the widely used 'copyleft' scheme)? To what extent does this FOSS 'authorship' deviate from the late 18th-century Romantic aesthetic that has purportedly shored up modern copyright law[1]? Have FOSS licensing schemes succeeded in carving out for FOSS programmers a unique legal persona that can be detached from the established software copyright?

Compared with many scholarly writings on the legal enforcement of FOSS licences, the size of the legal literature tackling the above questions about FOSS authorship is considerably smaller[2]. Dusollier observes that '[t]he author is barely mentioned in copyleft, despite playing a prominent role in the system' and this marked absence 'unfortunately conceals the importance of the author figure in the philosophical model of copyleft'.[3] As all copyleft licences are also copyright licences in the first place[4], Dusollier's observation tallies with Ginsburg's worry that 'the figure of the author is too-often absent' in 'contemporary debates over copyright' and this absence may only lead to an incomplete understanding of 'copyright's role in fostering creativity'.[5] In the similar vein, the lack of discussion of

---

1 For a definitive account of the Romantic-author vision and its lasting influence on modern copyright, see Martha Woodmansee, 'The Genius and the Copyright: Economic and Legal Conditions of the Emergence of the "Author" ' (1984) 17 *Eighteenth-Century Studies* 425-48

2 For example, Dusollier's attempt to link FOSS authorship with the postmodern aesthetic in a 2003 law journal article still remains arguably the most important contribution in the legal literature. Severine Dusollier, 'Open Source and Copyleft: Authorship Reconsidered?' (2003) 26 *Columbia Journal of Law and the Arts* 281-296. More recently, Mira Rajan has also provided her view on the issue. See, in particular, Section II (B) (subtitled 'Free Software: A Practical Need for Moral Right') in Rajan, 'Creative Commons: America's Moral Rights?' (2011) 21 *Fordham Intellectual Property and Entertainment Law Journal* 905-969, 936-45; Rajan, in an earlier article, has sketched out a few problems concerning programmers' legal authorship in a general context. Rajan, 'Moral Rights in Information Technology: A New Kind of "Personal Right"?' (2004) 12 (1) *International Journal of Law and Information Technology* 32-54, 48-49

3 Dusollier, ibid., 288

4 For example, the most widely used copyleft licence—GNU General Public Licence (GPL)—clearly recognises contributors' right to assert copyright in their contribution. Preamble, GPL 3.0 at <http://www.gnu.org/licenses/gpl.html> accessed 30 May 2013

5 Jane Ginsburg, 'The Concept of Authorship in Comparative Copyright Law' (2003) 52 *DePaul Law*

authorship in FOSS licensing schemes can also risk losing sight of the whole picture of the FOSS licensing jurisprudence.

The main thrust of this paper is that neither the Romantic-author vision nor the postmodern authorless creativity is suitable for defining FOSS programmers' authorial and legal persona(s). It is proposed that the issue can be better understood in the context of the computer hacker tradition[6] that has had a lasting influence in moulding FOSS programmers' authorial consciousness. Under this hacker tradition, FOSS programmers do not work as individualistic Romantic author, but they are practically minded 'craftsmen' who are steeped in software development as an engineering discipline. The creativity of these FOSS programmer-craftsmen is not just driven by a desire to *express* their unique individual genius personalities, but rather it is also sustained by an intrinsic enjoyment of making software as *functional* artefacts[7]. Their craftsmanship does not prevent them from claiming authorship, at both individual and collective levels, to their work via their licensing schemes. This re-examination will also call into question US copyright's treatment of software programs as literary works but not utilitarian objects and thus shed some light on the mechanism of FOSS licensing schemes that are in themselves a product of the hacking tradition.

In order to elaborate on the above argument, the rest of the paper is divided into three parts. The first part (Section 2) gives a brief review of the history of FOSS and how programmers' moral right of attribution is situated in the hacker tradition. It also introduces a significant legal lacuna in Anglo-American copyright law, which fails to recognise software programmers' moral right of attribution. The second part (Section 3) assesses the influence of the Romantic aesthetic in modern software copyright law. The Romantic author vision is then contrasted with two alternative theories of authorship, i.e., postmodernism (by Barthes and Foucault) and the 'craftsmanship' theory (by Richard Sennett[8]), the latter of which I argue is more appropriate to describe FOSS programmers' practical mode of creativity but has been unfairly neglected in the legal literature. The third part (Section 4) explores programmers' legal persona as shaped by FOSS licensing schemes, which attempt to create a private regime of moral right of attribution for programmers. It spells out the relationship between FOSS programmers' 'attribution' right and the craftsmanship theory, which deviates from the mainstream doctrinal understanding of 'authorship as property' under US case law. The fourth part (Section 5) concludes.

Review 1063-92, 1063

6 For a historical account of the early hacker culture, Steven Levy, *Hackers—Heroes of the Computer Revolution* (Penguin Books, London 1984, 1994)

7 Software is both functional and textual, but current copyright law tends to emphasises on the textual aspect of programming. Martin Kretschmer, 'Software as Text and Machine: The Legal Capture of Digital Innovation', 2003 (1) *The Journal of Information, Law and Technology* (JILT) at <http://www2.warwick.ac.uk/fac/soc/law/elj/jilt/2003_1/kretschmer/> accessed 30 May 2013

8 Richard Sennett, *The Craftsman* (Yale University Press, New Haven & London 2008)

2. A 'Private' Moral Right Regime?

This section gives a brief historical overview of the development of FOSS licensing. It highlights the conspicuous absence of moral rights protection of software programmers in the Anglo-American copyright system. In order to fill this significant legal lacuna, various FOSS licensing schemes have incorporated an attribution requirement, whose prevalence has arguably created a 'private' moral right regime as well as a difficult puzzle to figure out about FOSS programmers' legal persona.

2.1 From the Hacker Ethic to Open Source: A Very Brief History

The FOSS movement emerged from the computer-hacker community which was originally based in a few US academic institutions such as the Massachusetts Institute of Technology (MIT) in the 1950s and 1960s. Early computer hackers dutifully observed their community norm known as the 'hacker ethic' according to which they were 'actively willing to share technical tricks, software, and (where possible) computing resources with other hackers'.[9] This hacker ethic was carried out in full measure well into the early 1970s. Richard Stallman, who later founded the free software movement, recalls that when he first joined the MIT Artificial Intelligence Lab in 1971, he naturally 'became part of a software-sharing community that had existed for many years.' This software-sharing ethic, according to him, is 'as old as computers, just as sharing of recipes is as old as cooking.'[10]

However, in the early 1980s, the rise of copyright control over software[11] soon eclipsed this software-sharing hacker ethic and a lot of former hackers were hired by proprietary software corporations.[12] Deeply disappointed by this shift, Stallman invented the very first 'copyleft' licence in 1985 after a two-year long dispute with James Gosling, who sold his version of Emacs to a proprietary software company.[13] Note that this 1985 version of copyleft was written solely for the GNU Emacs programming editor. It was not until four years later that

---

[9] Eric Raymond, 'Hacker Ethic' in *The New Hacker's Dictionary* at <http://www.catb.org/jargon/html/H/hacker-ethic.html> accessed 30 May 2013

[10] Stallman, 'The GNU Operating System and the Free Software Movement' in Chris DiBona, Sam Ockman & Mark Stone (eds) *Open Sources: Voices from the Open Source Revolution* (O'Reilly, Sebastopol 1999) 53-70, 53

[11] In 1980, the US Congress officially extended its copyright law protection to cover software programs. This is largely based on Melville Nimmer's recommendation that software can be likened to literary works where copyright should subsist. See Anthony Clapes, Patrick Lynch, and Mark R. Steinberg, 'Silicon Epics and Binary Bards: Determining the Proper Scope of Copyright Protection for Computer Programs' (1987) 34 *UCLA Law Review* 1493-1594

[12] In around 1983, Stallman witnessed the gradual decline of the hacker ethic in the MIT AI Lab due to the propertisation of software. Levy, supra note 6, 419-427

[13] Christopher Kelty, *Two Bits—The Cultural Significance of Free Software* (Duke University Press, Durham 2008) 188-199

Stallman turned this Emacs-specific licence into a generic licence known as the GNU General Public License (GPL) that can be used for any free software project.[14] The GPL guarantees its users four kinds of 'software freedom' as listed by the Free Software Definition:

- The freedom to run the program, for any purpose.
- The freedom to study how the program works, and adapt it to your needs.
- The freedom to redistribute copies so you can help your neighbour.
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.[15]

The attempt to save and revive the hacker culture by using GNU GPL later went beyond Stallman's own GNU project. In the early 1990s, Stallman was invited to give a public speech about free software in Finland. In the audience was Linus Torvalds, who was then a college student from Helsinki University.[16] Torvalds was very intrigued by the idea of copyleft and when he released his fledgling Linux kernel program, he did not hesitate to license it under the GPL, which enjoined all contributors to share their contributions with the project. The GPL created a snowballing effect in the growth of the kernel, which integrated a huge number of contributions into one legally compatible collective work. It is noteworthy that Linux is not the whole operating system but only a kernel. It is surrounded by many non-kernel user-space programs to form an entire workable operating system.[17] Stallman's own GNU project has produced many widely used non-kernel programs, but its attempt to produce its own kernel known as 'Hurd' has not been successful. As the Linux kernel filled nicely into this gap in the GNU project, Stallman insisted that the whole operating system should be named 'GNU/Linux' instead of just 'Linux'. This naming controversy is a good example showing that FOSS programmers care deeply about authorial attribution concerning their work.[18]

In 1998, the success of 'GNU/Linux' further inspired Eric Raymond—an ambitious hacker who disagreed with Stallman's 'free software' puritanism—to coin the term 'open source', which signalled a new determination to integrate non-proprietary software into the

---

[14] This is GNU GPL 1.0 (1989) followed by GPL 2.0 (1991) and GPL 3.0 (2007)

[15] Richard Stallman, *The Free Software Definition* (2013) at <http://www.gnu.org/philosophy/free-sw.html> accessed 30 May 2013

[16] Linus Torvalds and David Diamond, *Just for Fun—The Story of an Accidental Revolutionary* (HarperBusiness, New York 2001) 58

[17] The kernel is the innermost part, i.e., the core, of an operating system. Applications such as the compilers are user-space utilities that surround the kernel. See Ellen Siever, Stephen Figgins and Robert Love, Arnold Robbins, *Linux in a Nutshell*, 6th ed. (O'Reilly, Sebastopol 2009) p.1; Robert Love, *Linux Kernel Development*, 3rd ed. (Addison-Wesley, Upper Saddle River 2010) p.4

[18] Richard Stallman, 'What's in a Name?' (2007) at <http://www.gnu.org/gnu/why-gnu-linux.html> accessed 30 May 2013

commercial mainstream.[19] 'Open source' is a more business-friendly approach that asserts itself to be a software development methodology superior to its proprietary counterpart. It contains a detailed list of requirements known as the 'Open Source Definitions' (OSD) that all open source licensing schemes should conform to. The OSD not only addresses the issues that have already been mentioned in the Free Software Definition, but it also touches upon programmers' reputation management in relation to their code. Section 4 of the OSD stresses the necessity to protect the 'integrity of the author's source code'. An open source licence 'may require derived works to carry a different name or version number from the original software'[20] in order to distinguish the original code from the modified one. This is based on the rationale that open-source licences should play a role in protecting the reputation of the 'author' of the relevant code:

> Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.[21]

2.2 Filling the Lacuna: The Prevalent Authorial Attribution Requirement

In the spirit of Section 4 of the OSD, almost all FOSS licences require downstream distributors to retain the attribution information about the original contributors in all future public redistributions. Legal scholars have been well aware that there is a strong norm of attribution in the FOSS community, where licences are employed to make sure that credit goes to the right source. Fisk observes that '[a]ttribution is important to many participants in the open source movement, even though exclusivity is shunned'.[22] In a similar vein, the *Free Software Act* (FSA) as proposed by the Free Software Consortium summarises the licensing norm of attribution in three points:

- The author of any free software program retains the right of attribution to his/her work.
- Any modifier must acknowledge the authorship of the original program and the authorship of the modification.
- All authorship must always be correctly attributed.[23]

---

[19] Eric Raymond, *The Cathedral and the Bazaar*, 2000, version 3.0 at <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/> accessed 30 May 2013

[20] Section 4, OSD at <http://opensource.org/osd-annotated> accessed 30 May 2013

[21] Rationale of Section 4 of the OSD, ibid.

[22] Catherine Fisk, 'Credit Where It's Due: The Law and Norms of Attribution' (2006) 95 *Georgetown Law Journal* 49-117, 89

[23] Jaco Aizenman, Maureen O'Sullivan, Martin Pedersen, Pedro Rezende, Shilu Shah, Pia Smith and Jorge Villa, *Free Software Act (Draft)* (2004) 1 (4) *SCRIPT-ed* <http://www.law.ed.ac.uk/ahrc/script-ed/issue4/FS-Act.pdf> accessed 30 May 2013

The above list registers an obvious desire to build a fair and effective attribution system among FOSS programmers. If fully realised, it would amount to a regime akin to the Berne Convention's moral right of 'attribution', which is also known as authors' right of 'paternity'. Strictly speaking, authors' paternity right is independent from authors' economic right of property, but it is a type of personality right. Article 6*bis* of the Berne Convention makes it clear that the paternity right is distinguishable from 'author's economic rights, and even after transfer of the said rights, the author shall have the right to claim authorship of the work […].'[24] On top of the paternity right, the same Article 6*bis* also names another non-economic moral right 'to object to any distortion, mutilation or other modification of, or other derogatory action in relation to, the said work, which would be prejudicial to his honour or reputation'[25]. This right is commonly known as the author's right of 'integrity'.

One should not confuse the Berne Convention's right of 'integrity' with the meaning of the word in the phrase 'Integrity of The Author's Source Code' as referred to in Section 4 of the Open Source Definitions (OSD). Although the term 'integrity' is used on both occasions, the scopes of the two are not exactly the same. Under the Berne Convention, authors' right of integrity is for preventing the authorial work from being distorted, mutilated or derogatorily modified. In contrast, FOSS licensing compatible with the OSD gives users the software freedom to modify the code in any manner, which may even include 'distortion' or 'mutilation' of the code, so long as the 'distorted' or 'mutilated' code is not misattributed to the original programmers.[26] Conversely, if a follow-up programmer makes good (rather than derogatory) modification of original code, FOSS licensing will also make sure this improvement is not attributed to the original programmers either. As Fisk observes: 'Although the explanation of the attribution requirements contained in the licenses are more focused on preventing wrongful attributions of blame than credit, presumably if a modification proves to be wonderful, the original authors will not get credit either.'[27] In this sense, Section 4 of the OSD regarding 'Integrity of The Author's Source Code' is really about a right *against false attribution*, which is closer to the 'paternity' right than the 'integrity' right under the Berne Convention.

Unfortunately, the paternity right under the Berne Convention is not directly applicable to software programmers, according to Anglo-American copyright law. This has created a significant lacuna that seems to be in need of being filled by FOSS licensing as a private arrangement of attribution.[28] In the US, only visual artists but not computer programmers are

---

[24] *Berne Convention for the Protection of Literary and Artistic Works* (1971 revision with 1979 amendments)

[25] Ibid.

[26] Bruce Perens, 'Open Source Definition' in Chris DiBona, Sam Ockman & Mark Stone (eds) *Open Sources: Voices from the Open Source Revolution* (O'Reilly & Associates, Sebastopol 1999) 171-188, 178

[27] Fisk, supra note 22 at 90

[28] At the moment, contractual arrangements seem to be the only basis of attribution in the Anglo-American context, as statutory law excludes programmers from moral rights protection on both sides of the Atlantic.

entitled to the moral right of attribution.[29] In the UK, computer programmers are expressly excluded from having the right to be identified as author[30], and this attribution right is only conferred to some categories of non-programming creators who affirmatively assert their attributional interest.[31] However, the British copyright law traditionally gives authors a right against 'false attribution', which may still be applicable to computer programmers. This British indigenous moral right is not derived from the Berne Convention, but it harks back to the UK Fine Arts Copyright Act 1862, and has its reincarnations respectively in Section 43 of the Copyright Act 1956 and Section 84 of CDPA 1988.[32] Lai argues that this right against false attribution is a historical 'anomaly' and that it makes little sense for computer programmers to have it without having the right of attribution in the first place.[33] In comparison, US programmers do not readily have a similar right against false authorial attribution under their copyright law, but they may have an analogue trademark device to protect the authorial origin under the common law action of 'passing off' as codified in Section 43(a) of the Lanham Act.[34] In 2001, the Lanham Act was successfully used to protect the authorial origin of a FOSS project known as 'Coolmail'.[35] However, two years later, the protection of the paternity right under the Lanham Act was put to an end by a US Supreme Court decision, which ruled that trademark law should not function as a kind of 'mutant copyright' to protect authorial attribution in *Dastar v Twentieth Century Fox.*[36] In short, although software programmers are classified as 'literary' authors under Anglo-American copyright law, their moral rights are not fully recognised. In this sense, programmers may be seen as a group of second-class authors under present law.

To summarise, the prevalent attribution requirement of attribution in FOSS licensing, following the spirit of Section 4 of the OSD, has effectively mandated a *limited*[37] private regime of software programmers' moral right of attribution that has not been recognised by the US copyright legislation. By doing so, open-source licences pose a real conundrum for lawyers to grapple with: do these licences really subvert the mainstream proprietary culture of software ownership or do they simply strengthen this culture by adding an extra layer of

---

29 Visual Artists Right Act 17 U.S.C s.106A ,

30 CDPA Section 79 (2) (a)

31 CDPA Section 77

32 Hugh Laddie, Peter Prescott, Mary Vitoria, Adrian Speck & Lindsay Lane, *The Modern Law of Copyright and Designs* (Butterworth, London, Edinburgh & Dublin 2000) 585-6

33 Stanley Lai, *The Copyright Protection of Computer Software in the United Kingdom* (Oxford & Portland, Oregon: Hart Publishing, 2000), 20

34 Section 43 (a) of Lanham Act is mainly used against misrepresenting the commercial origin of goods and service. However, it has also been successfully used to protect authors' moral rights in the US. See, for example, *Gilliam v ABC* 538 F.2d 14 (2d Cir.1976); *Follett v New American Library* 497 F. Supp. 304 (SDNY, 1980)

35 *Planetary Motion v. Techsplosion* 261 F.3d 1188 (11th Cir.2001)

36 539 US 23 (2003), 34

37 This private moral right regime is *limited* in the sense that it only intends to protect FOSS programmers' authorial attribution, but not their artistic integrity.

attribution right protection? The following analysis tries to show that the attribution clause in FOSS licensing is largely a makeshift solution or a 'hack' into the existing copyright law. Far from a complete overhaul of the law, this solution can be likened to a kind of software 'patch' created to fix a particular problem in a buggy system rather than devising an alternative solution from scratch. In this light, the attribution clause does not really weaken current copyright law, but the enforceability of the former relies on the latter. In fact, it will be shown later that, in judicial practice, the FOSS programmers' attribution right tends to be aligned with copyright owners' pecuniary interests as represented in the landmark ruling *Jacobsen v Katzer*.[38] In Section 4 of this article, I will offer a detailed critique of the *Jacobsen* ruling from the vantage point of the craftsmanship theory, which proposes a moral right of attribution detached from the proprietary copyright system. Before unravelling this critique, I need to explain what is meant by 'craftsmanship' as opposed to the Romantic and postmodern aesthetics.

3. Rehabilitating Craftsmanship: Authorial Personas of FOSS Programmers

This section examines three authorial personas of FOSS programmers, under Romanticism, postmodernism and the craftsmanship theory, respectively. It first traces the influence of the idea of the Romantic author in software copyright law. This is then followed by a brief account of the postmodern critique, which declares the 'death' of the author. The analysis will show that FOSS authorship is driven neither solely by Romanticism nor postmodernism, but that it can be better understood under the authorial persona of 'craftsman', whose practical mode of problem finding and problem solving is completely in line with the indigenous hacker tradition of FOSS programmers.

3.1 The Lingering Romantic Aesthetic in Software Copyright

The individual 'author', who can be credited as the sole originator of a creative work, is a construct of relatively recent pedigree. It largely stems from the Romantic movement since the late eighteenth century when literary writers were elevated to the position of self-inspired 'genius'[39]. This elevation emphasised the ability of an individual writer who could derive inspiration from his inner mind rather than an external source such as God or a muse.

---

[38] 535 F.3d 1373 (Fed. Cir. 2008)

[39] Romantic authorship can also be seen as an aesthetical façade of authors' legally enforceable moral rights as opposed to their economic rights. Historically, the Romantic vision plays a role in justifying and propelling the legal protection of authors' autonomy in self-expression, however unsatisfactory this justification may be. See Christopher Aide, 'A More Comprehensive Soul: Romantic Conceptions of Authorship and the Copyright Doctrine of Moral Right' (1990) 48 *University of Toronto Faculty of Law Review* 211-28. For the historical context of Romantic authorship, see also, the discussion of the 'authorship norm' as opposed to the 'marketplace norm' by Paul Edward Geller, 'Must Copyright Be For Ever Caught Between Marketplace and Authorship Norms?" in Alain Strowel (ed.) *Of Authors and Origins: Essays on Copyright Law* (Clarendon Press, Oxford 1994) 159-201

Woodmansee argues that the rise of Romanticism comes with the suppression of the non-imaginative 'craftsmanship' element in the creative process, whereby the literary writer produces a special kind of private 'property' based on the expression of his unique personality.

> [Romantic writers] minimized the element of craftsmanship (in some instances they simply discarded it) in favor of inspiration, and they internalized the source of that inspiration. That is, the inspiration for a work came to be regarded as emanating not from outside or above, but from within the writer himself. 'Inspiration' came to be explicated in terms of *original genius*, with the consequence that the inspired work was made peculiarly and distinctively the product—and the property—of the writer.[40] (original emphasis)

The literary writer's rise from 'craftsman' to 'author-genius' is sometimes likened to a shift from 'mirror' to 'lamp' as a consequence of the Romantic movement.[41] The pre-Romantic craftsman is like a mirror that merely reflects the external world, while the Romantic author is like a lamp that emits creation like a source of light. A Romantic author-genius is distinguished by his ability to generate 'original' creation *ex nihilo* as exemplified by Wordsworth's testimony that '[g]enius is the introduction of a new element into the intellectual universe […]'.[42] The growing eminence of the Romantic 'cult of the genius'[43] also had its repercussion in the development of modern copyright law[44]. It is argued that that today's copyright regime is precisely built upon the cult of the Wordsworthian author-genius:

> Our laws of intellectual property are rooted in the century-long reconceptualization of the creative process which culminated in high Romantic pronouncements like Wordsworth's to the effect that this process *ought* to be solitary, or individual, and introduce 'a new element into the intellectual universe.' Both Anglo-American 'copyright' and Continental 'authors' rights' achieve their modern form in this critical ferment, and today a piece of writing or other creative product may claim legal

---

40 Woodmansee, supra note 1 at 427

41 Meyer Abrams, *The Mirror and the Lamp: Romantic Theory and Critical Tradition* (Oxford University Press, Oxford 1971)

42 William Wordsworth, 'Essay, Supplementary to the Preface', quoted in Woodmansee, 'On the Author Effect: Recovering Collectivity' (1992) 10 *Cardozo Arts and Entertainment Law Journal* 279-92, 280

43 Tim Blanning, *The Romantic Revolution* (Orion Books, London 2010) 31-36

44 It is worth remembering that the birth of modern copyright, which is conventionally marked by the Statute of Anne of 1709, happens in the age of enlightenment, which precedes the Romantic era. Geller points out that copyright has historically been caught between the enlightenment value (also known as the 'marketplace norm') concerning the dissemination of knowledge and the Romantic authorship norm concerning the authorial autonomy of self-expression. Geller, supra note 39

The relatively short history of FOSS licensing also reflects this struggle. The licensing terms that allow re-use and re-mix embodies the enlightenment value, while the attribution clause has largely reified the authorship norm.

> protection only insofar as it is determined to be a unique, original product of the intellection of a unique individual (or identifiable individuals).[45] (original emphasis)

Now zoom in onto the more specific area of software copyright, where there has been no shortage of academic works that bear out Woodmansee's worry about copyright law's uncritical acceptance of literary Romanticism. Jaszi, a champion of Woodmansee's thesis, observes that 'lawyers and judges have invoked the vision of the Romantic "author-genius" in rationalizing the extension of copyright protection to computer software', because software programs are 'no less inspired than traditional literal works, and that the imaginative processes of the programmer are analogous to those of the literary "author".'[46] It is noteworthy that the main source that Jaszi relies upon to make his observation comes from an earlier article entitled 'Silicon Epics and Binary Bards' written by a team of IBM lawyers in 1987[47]. 'Silicon Epics and Binary Bards' straightforwardly likens software to the 'epic poetry of the Information Age'[48] and a programmer is correspondingly the 'poet' of his imaginative creation. Note that this programmer-as-poet vision is not preached for the first time either, but is derived from Frederick Brooks's classical 1975 work on software design, which forcefully articulates a Romantic vision about software programming as a creative process coming out of a poetic programmer's imagination:

> The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures[…][49]

Under this logic, software programming is by no means a mindless job but involves a programmer-poet's active 'exertion of the imagination' that impresses his unique personality into the code.[50] This view resonates strongly with the rationale behind the legislative extension of US copyright law to cover software in 1980 based on the copyright scholar Melville Nimmer's recommendation to the US National Commission on New Technological Uses of Copyrighted Works (CONTU).[51]

It is worth noting that the influence of Romanticism has already been eroded in current copyright. It is difficult to have the Romantic persona to account for the whole picture of

---

45 Woodmansee, supra note 42, 291-2

46 Peter Jaszi, 'On the Author Effect: Contemporary Copyright and Collective Creativity' (1992) 10 *Cardozo Arts and Entertainment Law Journal* 293-320, 297-8

47 Clapes, Lynch and Steinberg, supra note 11

48 Ibid., 1584

49 Brooks, *Mythical Man-Month*, 7-8, quoted in ibid., 1497

50 Arthur R. Miller, 'Copyright Protection for Computer Programs, Databases, and Computer-Generated Works: Is Anything New Since CONTU' (1993) 106 *Harvard Law Review* 977-1073, 983-4

51 Clapes, Lynch and Steinberg, supra note 11, 1583

copyright doctrines, such as issues dealing with the actual length of copyright duration, collective works, limitations and exceptions.[52] More specifically, the Romantic-author vision that treats software as the poetic expression of a programmer's original personality can be subject to at least two strands of criticism. First, the Romantic view derives from an author-centred perspective where the programming author-genius is deemed as the sole source of creation. It profoundly ignores the other side of the equation which is software *users*' contribution to a software product. Especially in a FOSS environment, this problem becomes quite obvious because all users are potential 'co-developers' who can participate in the collaborative programming process.[53] Second, when the Romantic view is skewed one-sidedly towards the analogy that software programming is a literary *expressive* activity, it ignores the fact that programming is also an engineering discipline that involves making *functional* objects. In order to counterbalance this bias, it is also important to see software programmers as practical 'craftsmen' who make workable things that do not necessarily have to be a vehicle of expressing programmers' personalities. As these two strands of criticism are hugely important to form a panoramic picture of FOSS authorship, I will deal with the first one in the context of the *postmodern* critique of Romanticism and elaborate in some detail on the second one in relation to the *craftsmanship* model.

3.2 The Postmodern Critique of the Romantic Author

Postmodernist critics do not see the author as an authoritative figure who can exert a total and despotic control over a creative work. Instead, the putative 'author' fades away after the creative process is instigated by its initiating creator. A postmodern literary work is in fact a *discourse in progress* and its future development is dependent on the participation of its readers. Barthes in his 1967 essay famously declared the 'death' of the original author. This metaphoric death reduces the 'author' to the bare status of a 'scriptor' who merely scribbles

---

[52] For a detailed evaluation of the lingering influence of Romanticism in US copyright, see Mark Lemley, 'Romantic Authorship and the Rhetoric of Property' (1997) 75 *Texas Law Review* 873-906

In particular, the declining influence of Romanticism is also shown in its presence in copyright's dealing with collaborative works with multiple authors, which certainly goes beyond the Romantic solitary author model. It is worth noting that different terminologies are used by the UK and US copyright systems to describe collective works. In the UK, copyright subsists in 'compilation' (s3(1)a, CDPA 1988) as a type of collective *undivided* 'literary work'. Legal scholars such as Bainbridge believes that software is exactly such copyrightable 'compilation' comprising many *undivided* components rather than a 'database' comprising *separable* individual components under the UK law. In the US context, the term 'compilation' has a slightly different meaning, because it also covers 'collective works' comprising separable works. 17 USC 101. In the case of FOSS, I am inclined to agree with Bainbridge, who argues that software is better seen as an *undivided* collective work (or 'compilation' under the UK law), and I will show later in this article that the job of those FOSS project leaders is exactly to aggregate individually contributed code into undivided whole that can run on its own two feet. Bainbridge, *Legal Protection of Computer Software* (Tottel Publishing, Heywards Heath 2008) 67

[53] Raymond, supra note 19

down the text but claims no interpretive authority over the text.[54] Two years later, Michel Foucault, in an equally celebrated essay 'What Is an Author?', indicates that the 'author' in question may not be an irreversibly 'dead' corpse. Instead, the authorial death sentence is commuted to the so-called 'author function' which is employed as the 'principle of thrift in the proliferation of meaning'.[55] A Foucauldian author relinquishes the total control over the work, but he still plays a role in restricting the uncontrolled and free-flowing interpretive process that can be participated in by readers. This thrifty non-proliferation 'author' is better seen as an instigator or founder of a discursive activity that may be further shaped by participation of readers and he shall have no monopoly on determining the final shape of the collectively produced creation.

Despite the subtle difference between Barthes and Foucault, both theorists articulate a *reader*-centred perspective of literary creation in stark contrast to the *author*-centred Romanticism. Most interestingly, this reader-centred perspective has already radiated out into theorisation about FOSS production, where software 'users' take the place of literary 'readers'. The legal scholar Dusollier argues that the practice of FOSS licensing has largely fleshed out Foucault's postmodern production of the collective work as an ongoing discourse between software developers and users. These FOSS developers are 'authors', who consciously choose to use FOSS licensing schemes (especially copyleft) in order to give software freedom to their users. By doing so, they are effectively 'authoring' a Foucauldian collective work in the manner of conducting an open-ended discourse that allows contributions from users:

> The author is not only the initial founder of a discourse and instigator of a creation of which her contribution is only the first stage. She is also the figure by whom the whole of the collective creation finds itself marked by the stamp of freedom. In the chain of contributions, of works which will come to add incrementally to the first act, none will be able to escape the refusal of intellectual property rights exerted in a proprietary and exclusive manner. Foucault's desire for greater cultural freedom is brought to life in copyleft.[56]

Furthermore, the use of FOSS licences not only marks a shift of focus from Romantic 'author' to software 'user', but it also signals a different understanding of the concept of the authorial 'work'. Under the Romantic aesthetic, a work published by the author tends to be a finished product that is solely produced and fully owned by its creator. In other words, this Romantic authorial work is an objectification of an author's private labour, which may be

---

[54] Roland Barthes, 'The Death of the Author' in *Image, Music and Text* trans. by Stephen Heath (Fontana, London 1977) 142-148, 145

[55] Michel Foucault, 'What is an Author' in Josue E. Harari (ed), *Textual Strategies: Perspectives in Post-Structuralist Criticism* (Cornell University Press, Ithaca 1979) 141-160, 159

[56] Dusollier, supra note 2, 294-5

traded as alienable commercial property on the market.[57] In contrast, the postmodern aesthetic suggests that a 'work' is always a piece of work in progress that can be understood as an ongoing discourse. The postmodern work is started but not finished by its inaugural creator and it welcomes readers or users to make their contribution. So it is mostly likely to be a collective work that keeps evolving and expanding for an indefinitely long duration. As a consequence, it is much more difficult to commercialise a piece of ever-expanding postmodern work due to its lack of the sole author-ownership and discrete boundary.

3.3 Engaging with the Code: Craftsmanship and FOSS Programming

Compared with the author-centred Romanticism and the reader/user-centred postmodernism, the study of FOSS programmers as 'craftsmen' shifts the focus further to the *work in itself*. Here the craftsmanship 'work' does not just mean the literary *expressive* 'work' as dealt with by Romanticism, but it also refers to software as *functional* objects.[58] The craftsmanship model urges researchers to see programming not just as a literary activity but also as a practical craft. Craftsmanship in FOSS programming is an important but understudied phenomenon. In fact, it was not until the recent publication of the sociologist Richard Sennett's seminal book *The Craftsman*[59] that the connection between FOSS programming and craftsmanship was rendered clear. Sennett, by searching a long historical development of craftsmanship since the Homeric hymn to Hephaestus (master god of craftsmen), finds that FOSS programming as represented by the Linux kernel project is a prime example of the modern-day work-centred craftsmanship: Programmers 'who participate in "open source" computer software, particularly in the Linux operating system, are craftsmen who embody some of the elements first celebrated in the hymn to Hephaestus'.[60] This is because Linux embodies craftsmen's dedication to *the quality of the work in itself*, or in Sennett's own words, it is 'focused on achieving quality, on doing good work, which is the craftsman's primordial mark of identity.'[61]

3.3.1 Two Traits of Programmer as Craftsman

Programming as a work-centred practical craft has two salient traits that have been ignored by Romanticism. The first trait of craftsmanship challenges a deeply entrenched bias that privileges 'having ideas' over 'making objects' in Western society.[62] Sennett observes that

---

57 Peter Jaszi, 'Toward a Theory of Copyright: The Metamorphosis of "Authorship" ' (1991) 2 *Duke Law Review* 455-502, 471-480

58 Recall that software is both textual (as in human-readable source code) and machine-like (as in machine-readable object code). See Kretschmer, supra note 7

59 Sennett, supra note 8

60 Ibid., 24

61 Ibid., 25

62 Peter Dormer, 'The Status of Craft', in Dormer (ed.) *The Culture of Craft* (Manchester University Press, Manchester 1997) 18-19, 18

the historical trend 'has drawn fault lines dividing practice and theory, technique and expression, craftsman and artist, maker and user'.[63] Countering this trend, craftsmanship is against the arbitrary divide between the 'high' creative activity as conceiving original ideas and the 'low' creative activity as merely implementing those ideas. It is an attempt to reconnect 'hand' with 'head' by building 'a dialogue between concrete practices and thinking'[64] and thus rehabilitate the craftsmanship element that has been discarded by the Romantic movement.[65] In the context of FOSS, the craftsmanship argument is in line with Steven Weber's observation that FOSS is 'first and foremost an engineering culture—bottom up, pragmatic, and grounded heavily in experience rather than theory.'[66] This argument, when fully spelt out, may also provide useful ammunition to support some legal scholars' proposal to replace the generic copyright law with a *sui generis* regime for protecting software as functional objects.[67]

The second trait of FOSS craftsmanship is programmers' dedication to the quality of their work for its own sake. In Sennett's words, this dedication 'represents the special human condition of being engaged' and 'people become engaged practically but not necessarily instrumentally.'[68] The craftspeople's practical engagement, in the computer hacker tradition, simply means the use of the hacking skills to *take care of* the created work. Burrell Smith, an early designer of the Macintosh computer, comments that computer hacking is not 'necessarily high tech' but 'it has to do with craftsmanship and caring about what you're doing.'[69] The hackers' commitment to engage with or care about their creation does not result in the total *ownership* of the created work. Instead, it is more a matter of taking *stewardship* responsibility for software, which needs to be taken care of like a living object. In this sense,

---

63 Sennett, supra note 8, 11

64 Sennett urges readers to see the connection between hand and head in craftsmanship: 'Every good craftsman conducts a dialogue between concrete practices and thinking; this dialogue evolves into sustaining habits, and these habits establish a rhythm between problem solving and problem finding.' Ibid., 9

65 The Romanticist prejudice of 'head' over 'hand' is not entirely unfamiliar to copyright lawyers. It has its reincarnation in one of modern copyright law's authorship principles, which stipulates that 'authorship places mind over muscle'. Ginsburg, supra note 5, 1072

66 Steven Weber, *The Success of Open Source* (Harvard University Press, Cambridge, Mass. 2004) 164

67 Pamela Samuelson, Randall Davis, Mitchell D. Kapor, and J. H. Reichman, 'A Manifesto Concerning the Legal Protection of Computer Programs', (1994) 94 (8) *Columbia Law Review* 2308-2431

The recognition of the 'hands on' functional element in software creativity does not equal a support of 'software patent' or 'software-related invention patent'. FOSS programmers (such as Richard Stallman) have been hostile to patenting software. This is because patent offers a much stronger monopoly than copyright, which still allows for many user/reader exceptions. The craftsmanship argument only seeks to accurately describe the dual nature of software, which is both expression and function. It is merely a reaction against current copyright's unsatisfactory classification of software as a type of 'literary works', which arbitrarily separate expressions from functions in software development.

68 Sennett, supra note 8, 20

69 Levy, supra note 6, 434

a few leading FOSS programmers have already made a distinction between 'ownership' and 'stewardship' in relation to their creation: 'Ownership is something that is fully transferable from one owner to another without loss of values. [….] Stewardship**,** on the other hand, applies when something undergoes change, when it evolves, or when it has some kind of life cycle.'[70] Most importantly, a carefully stewarded FOSS project is often cared by a small group of lead programmers (such as Torvalds for the Linux Kernel), and it tends to nurture a long-term collaborative relation. The craftsmen's care by a core stewarding group of dedicated lead programmers is exactly the reason behind the longevity of many FOSS projects.[71]

Furthermore, FOSS programmers' stewardship responsibility to care about the quality of their work is also in line with computer hackers' meritocratic tradition, in which a programmer's merit is evaluated purely by the quality of his work rather than his personal attributes or social status. It follows the hacker ethic stipulating that '[h]ackers should be judged by their hacking, not bogus criteria such as degrees, age, race or position.'[72] This meritocratic principle shows that the hacker ethic as a work-centred ethic is almost the opposite of the creators' personality cult. Sennett observes that craftsmanship's focus on the quality of work has an 'impersonal character', which can be rather 'unforgiving', but it is well present in the Linux community.[73] This programmer-craftsmen's favour of work's quality over programmers' personality is also corroborated by Eric Raymond's observation of the 'strict meritocracy' in the hacker community, where 'the best craftsmanship wins' by the quality of the code in itself:

> In the hacker community […] one's work is one's statement. There is a very strict meritocracy (the best craftsmanship wins) and there's a strong ethos that quality should (indeed must) be left to speak for itself. The best brag is code that 'just works', and that any competent programmer can see is good stuff. Thus, the hacker culture's knowledge base increases rapidly.[74]

3.3.2 Differences between Craftsmanship and Postmodernism

On the surface, the above two traits of craftsmanship seem to share a similar postmodernist urge to deconstruct the Romantic cult of genius. However, a closer scrutiny reveals that the craftsmanship perspective is also subtly different from the postmodernist critique of the Romantic author in two aspects. The first is about individual programmers' *motivation* to

---

70 Chris DiBona, Danese Cooper, and Mark Stone, 'Introduction' in DiBona, Cooper, and Stone (eds), *Open Sources 2.0*, (O'Reilly, Sebastopol 2006) xxxvii

71 Ibid., xxxviii

72 Levy, supra note 6, 43

73 Sennett, supra note 8, 27

74 Eric Raymond, 'Homesteading the Noosphere', (2002) at <*http://www.catb.org/~esr/writings/homesteading/homesteading/*> accessed 30 May 2013

contribute to a FOSS project, while the second deals with *coordination* of individually made contributions into a functional whole. I will unravel each of them to show that the craftsmanship model represents an attempt to reconstruct, more than deconstruct, the role of individual programmers in FOSS collaboration as a cooperative craft.

Firstly, for the postmodern critique, it is not entirely clear why individual rank-and-file FOSS contributors are *motivated* to contribute to a collaborative project. Postmodernists simply avoid asking what the motivational forces are behind individual programmers; they assume that the collective work is capable of organising itself automatically after an author's 'death'. (Understandably, there is no point in asking why a 'dead' author should be motivated.) In contrast, the craftsmanship model does attempt to provide an explanation about the motivation issue: programmer-craftsmen are *primarily* driven by their intrinsic satisfaction or pleasure of writing code for its own sake. Linus Torvalds famously names this intrinsic pleasure-driven motivation the 'Entertainment with the capital E' within the Linux community.[75] 'Entertainment' of code writing is believed to draw thousands of Linux programmers into the kernel project because 'Entertainment is something intrinsically interesting and challenging'.[76]

Torvalds' 'Entertainment' as motivation does not cover every type of recreational activities. Rather, he sets limits on what qualifies, indicating that the 'Entertainment' should be linked with craftsmanship *skills*, which may be improved or perfected in practice. Examples of this kind of *skill-based* 'Entertainment' can be found in 'chess' games, 'painting' or 'mental gymnastics involved in trying to explain the universe' and, of course, software programming[77]. Over the course of FOSS collaboration, Linux contributors are likely to improve their programming skills, and as a virtuous circle, when they become more skilled, they tend to derive more 'Entertainment' and thus do more programming.[78] It is thus appropriate to see FOSS programming as an intellectual sport, where programmers derive satisfaction from coding just as professional athletes enjoy their sports:

> A very complex project like Apache or the Linux kernel brings the satisfaction of the ultimate in intellectual exercise. Much like the rush a runner feels while running a race, a true programmer will feel this same rush after writing a perfect routine or tight piece of code. […] The point is that many programmers code because it is what they love to do, and in fact it is how they define their intellect. Without coding, a programmer feels less of

---

75 Linus Torvalds, 'What Make Hackers Tick? a.k.a. Linus's Law' as a prologue to *The Hacker Ethic and the Spirit of the Information Age*, by Pekka Himanen, (Random House, NY 2001) xvi

76 Ibid., xv

77 Ibid.

78 This virtuous circle is often known as Isaac Stern rule (named after the famous virtuoso violinist) in musicianship: 'the better your technique, the longer you can rehearse without becoming bored.' Sennett, supra note 8, 38

> a person, much like an athlete deprived of an opportunity to compete.[79]

The second aspect of FOSS craftsmanship that is different from postmodernism concerns the coordination issue. Although the postmodern critique correctly points out that FOSS is collectively created, it does not further specify how the individually made contributions are actually *integrated* into a functional whole. For this reason, the postmodern critique has been accused of creating an imagined heroic 'Romantic collective author', who is no different from the subject it intends to critique, i.e., the Romantic individual author.[80]

The craftsmanship model is different from postmodernism (or Romantic collectivism) in the sense that it does not assume that collective works can organise themselves. Instead, it believes that a core group of lead programmers' deliberate efforts are needed to coordinate individually contributed code into a functional work. The annual reports published by the Linux Foundation since 2008 have consistently shown that the Linux kernel project is coordinated or stewarded by a small number of lead developers (known as the subsystem maintainers), who play a crucial role in integrating a huge number of contributions into final releases. This team of maintainers are quite like academic journals' editors or peer-reviewers who act as communities' gatekeepers. They are responsible for vetting and testing all submitted patches, which may be either rejected or approved into the mainline kernel.[81] This review process reflects Linux's practical need for quality control of their collective work and it has little to do with the expression of individual programmers' unique personalities. Kelty illustrates how the Linux maintainers led by Torvalds do their daily job to coordinate individuals' programming virtuosity into a collective functional work:

> Almost all of the decisions made by Torvalds and lieutenants were of a single kind: whether or not to incorporate a piece of code submitted by a volunteer. Each such decision was technically complex: insert the code, recompile the kernel, test to see if it works or if it produces any bugs, decide whether it is worth keeping, issue a new version with a log of the changes that were made. Although the various official leaders were given the authority to make such changes, coordination was still technically informal. Since they were all working on the same complex technical object, one person (Torvalds) ultimately needed to verify a final version, containing all the subparts, in order to make sure that it worked without breaking.[82]

---

79 Chris DiBona, Sam Ockman & Mark Stone, 'Introduction' in DiBona, Ockman & Stone (eds), *Open Sources—Voices from the Open Source Revolution* (O'Reilly, Sebastopol 1999) 1-17, 13

80 Margaret Chong, 'The Romantic Collective Author' (2012) 14 (4) *Vanderbilt Journal of Entertainment and Technology Law* 829-49

81 Jonathan Corbet, Greg Kroah-Hartman, Amanda McPherson, *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*, March 2012, <go.linuxfoundation.org/who-writes-linux-2012> accessed 30 May 2013

82 Kelty, supra note 13, 220

It is not difficult to find that Torvalds and his fellow subsystem maintainers' review work hardly amounts to the level of Wordsworthian 'originality' that generates absolutely new things *ex nihilo*. Instead, their job is about testing and combining other people's contributions and thus much more mundane than expressing their own imaginative personalities. However, it is still nonetheless a job of critical importance for the purpose of making a functional software object that can technologically stand by itself. This responsibility to review reinforces the two traits of craftsmanship as mentioned above. First, the review process reflects the first trait of craftsmanship, which seeks to build a dialogue between concrete practice and thinking—between that of 'hand' and 'head'—in collaborative programming activities. To test and then possibly integrate a submitted patch into the final project is largely a practical exercise of trial and error. This is a process where individually composed code is vetted against a practical criterion: whether it *works* with the rest of the software system as a whole. To implement an idea[83] into the final *collective* software work is as significant as to conceive that initial idea at the *individual* level. In other words, the final review process is exactly where the practical 'hand' element in craftsmanship reaches parity with the rarefied 'head' element when a programming idea is first conceived.

Secondly, the coordination process that involves Torvalds and his fellow-reviewers is also a matter of quality control, which aims to weed out the bad code and retain the good one for the collective work. It reflects FOSS programmer-craftsmen's second trait, which is their dedication to the quality of the work for its own sake.[84] The way that FOSS programmers measure and monitor the quality of software patches for their project bears a strong resemblance to the peer review process in the scientific community, which also deeply cares about the quality of its work. It is in line with at least two of the norms identified in Robert Merton's scientific ethos.[85] One is the norm of 'universalism' and the other the norm of 'organised scepticism'. The former judges the merit of a piece of scientific work on 'preestablished impersonal criteria'[86], while the latter is 'a methodological and an institutional mandate'[87] necessary for verifying scientific claims' validity through the

---

83 US copyright excludes non-expressive elements such as 'ideas' from copyright protection. 17 USC 102(b). Its case law also provides a safety net by excluding those expressions that are inseparable from ideas under the 'merger' doctrine. *Baker v Selden* 101 U.S. 99 (1879); *Lex mark v Static Control Components*, 387 F.3d 522 (6th Cir. 2004). This merger doctrine is largely in line with the craftsmanship argument, which treats expressions and non-expressions (ideas or function) as inseparable in software development.

84 Here the coordination among programmers puts a further emphasis on dedication to the quality of the *collective* work as a whole, while the quality issue discussed earlier on is mainly about the work done by each *individual* craftsman.

85 For a detailed account of Merton's sociology of science, see Aaron L. Panofsky, 'A Critical Reconsideration of the Ethos and Autonomy of Science' in Craig Calhoun (ed), *Robert Merton: Sociology of Science and Sociology as Science* (Columbia University Press, New York 2010) 140-163

86 Robert K. Merton, *On Social Structure and Science*, ed. by Piotr Sztompka (The University of Chicago Press, Chicago & London 1996) 269

87 Ibid., 276

self-checking mechanism of peer review. The two norms point in the same direction of building a meritocratic system that focuses on the quality[88] of scientific work without relying on certain individuals' genius status.

To summarise, under the craftsmanship model, individual programmers are real human creators who are *motivated* by the intrinsic pleasure of coding. At the same time, a small group of lead programmers undertake extra responsibility to *coordinate* individually submitted code into a collective executable program. In contrast, a postmodern software project seems to disown its individual authors after the code is written, based on the assumption that individually contributed code may automatically aggregate into a functional whole. In fact, without the coordination by a core group of lead programmers, this kind of postmodern 'authorless' work may well be a collection of unrelated software fragments that would fall apart in the end. In short, the craftsmanship approach has at least two advantages over postmodernism in explaining FOSS authorship. First, it explains individual rank-and-file programmers' *motivation* as craftsmen's enjoyment of coding for its own sake. Postmodernism avoids dealing with this 'motivation' issue, while the craftsmanship theory tackles it head-on. Second, individually contributed lines of code cannot be used straight away, but it needs to be aggregated into a functional whole *coordinated* by lead FOSS programmers. Postmodernist thought seems to imply that creative works are self-organised without conscious efforts of coordination. The craftsmanship theory gives full recognition to the authorial role of a small group of lead programmers (such as Linus Torvalds or Richard Stallman) for their stewardship responsibility in coordinating a project as a whole. This coordination effort is perfectly in line with programmer-craftsmen's dedication to the quality of their work.

3.4 Intermediate Conclusion: Why Does Craftsmanship Matter?

The above discussion has surveyed three authorial personas that may be possibly assumed by FOSS programmers. The craftsmanship approach, which is championed by this article, has at least two layers of significance. First, it is more direct and accurate in describing FOSS programmers' authorial persona from their indigenous hacker tradition. It does not have to stretch the Romantic or postmodern aesthetic, which is originally developed in literary and artistic criticism. Second, the craftsmanship theory can also build a conceptual *bridge*, from programmers' 'authorial' persona, towards their corresponding 'legal' persona as constructed through FOSS licensing schemes. As the first layer has just been discussed above, I will now move onto the second layer, which deals with the legal persona marked by the prevalent authorial attribution licensing requirement in the following section.

4. Re-inventing the Legal Persona for FOSS Authorship

[88] I do not imply that quality is a standard for software protection. As will be shown in below Section 4, the craftsmanship persona, when translated into the licensing language, is only limited to the requirement of authorial attribution, and it does not set a quality threshold for copyright protection.

This section deals with programmers' legal persona as manifested through FOSS licensing schemes. FOSS programmers' legal persona would appear to be neither a direct translation of literary Romanticism nor that of the postmodern authorless creativity. On the one hand, it does not fit with the lone Romantic author because FOSS licensing does welcome users to modify and redistribute the original author's creation. On the other hand, postmodernism cannot explain the ubiquitous 'attribution' clause in FOSS licensing whereby licensees are required to credit the creators of the relevant code. Again, this attribution requirement seems to be better explained under the quality-centred craftsmanship model: in a meritocratic FOSS community, programmers are assessed by the quality of their work, while an attribution system plays precisely the necessary role in linking FOSS programmers' names with the relevant code. The craftsmanship theory will eventually lead to a normative call for a *limited* moral right regime recognising programmers' attribution right for its own sake, which should be detached from an economically minded proprietary copyright system.[89]

4.1 *Jacobsen v Katzer*: Failure to Depart from 'Authorship as Property'

To build an authorial link between programmers and code under FOSS licensing is important in two important aspects. First, it gives benefit, such as reputational gains, to programmers who contribute the code. Second, it shows programmers' willingness to take responsibility for making work that guarantees software freedom. These two aspects respectively correspond to two different schools of thought about the nature of authorial attribution. One treats attribution as an economic benefit derived from programmers' work as *private property*, while the other treats attribution as a badge of *authorial responsibility* for taking craftsmanship care of the code. In the following analysis, I use 'authorship as property' as a shorthand for the first school and 'authorship as responsibility' for the second. I will show that the landmark ruling in *Jacobsen v Katzer* is skewed towards 'authorship as property' but ignores 'authorship as responsibility', the latter of which can be equally crucial to the success of FOSS collaboration based on the craftsmanship model.

Although US copyright law has largely failed to reproduce a Berne-type attribution regime to protect programmers, this lacuna[90] may be filled by private property licensing schemes made by programmers wearing the legal persona of *copyright owner* of the software code. This has been essentially achieved, through FOSS licensing schemes, by having programmers' attribution right ride on the proprietary interests as owned by FOSS developers. Lastowka argues that US copyright only recognises authors' attribution right 'in a collateral fashion', which protects 'works of creative authorship as property' through the copyright licensing mechanism:

> It might be argued that copyright protects attribution in a collateral fashion. By

[89] Note that this article does not plead for a fully-fledged moral rights regime for computer programmers. The policy implication is limited only to FOSS developers' moral right of 'attribution' but not the right of 'integrity'.

[90] The legal lacuna has been explained in some detail in Section 2.2 of this article.

> *protecting works of creative authorship as property*, copyright enables the contractual protection of attribution. If an author can control the dissemination and reproduction of her work pursuant to copyright law, copyright law will grant her the contractual leverage to protect her attribution interests.[91] (emphasis added)

The above paragraph precisely articulates what I have called 'authorship-as-property' approach to programmers' paternity right under current US law. In other words, although attribution is not directly protected for its own sake under copyright legislation, it is disguised as a proprietary interest of the author-owner through a licensing scheme.

The possibility of securing collateral protection of authorial attribution as property has been recently vindicated in the landmark FOSS licensing case *Jacobsen v Katzer*.[92] This case involves a dispute over a FOSS project known as 'Java Model Railroad Interface' (JMRI) led by Professor Robert Jacobsen, who is a Berkeley physicist by profession and a model train hobbyist in his spare time. The JMRI code under dispute was then released under Artistic License (AL) 1.0. It is generally believed that AL has effectively created a private regime of *droit moral* enabling JMRI developers to have wider authorial control than allowed under the statutory language of the US copyright law. The Preamble of AL1.0 makes no effort to conceal this intent: 'The intent of [AL] is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of *artistic control* over the development of the package […]'.[93] (emphasis added) Fabricius comments that 'the essential novelty' of AL lies precisely in its 'granting the author more attribution and creative control than would be granted in the ordinary case of a copyright license to copy, distribute, and prepare derivative works'.[94] In this way, JMRI programmers are given 'a private moral right' that is akin to Section 106A of the US Visual Artists Right Act providing attribution right to certain visual artists.[95]

The actual dispute in *Jacobsen* revolves around a program called DecoderPro, which is a sub-project of the JMRI. In September 2006, the JMRI developers discovered that Matthew Katzer had copied and modified some DecoderPro files into his own proprietary product. At the same time Katzer deliberately removed the following information that would have identified JMRI contributors as authors of their code:

- the authors' names
- JMRI copyright notices

---

91 Greg Lastowka, 'The Trademark Function of Authorship' (2005) 85 *Boston University Law Review* 1171-1241, 1214

92 535 F.3d 1373 (Fed. Cir. 2008)

93 Preamble, Artistic License 1.0 at <http://opensource.org/licenses/Artistic-1.0> accessed 30 May 2013

94 Erich M. Fabricius, 'Jacobsen v. Katzer: Failure of the Artistic License and Repercussions for Open Source' (2008) *North Carolina Journal of Law & Technology* 65-88, 85

95 Ibid.

- references to the COPYING file
- and identification of SourceForge or JMRI as the original source of the definition files, and
- a description of how the files or computer code had been changed from the original source code.[96]

Katzer did not dispute his act of copying, but he contended that non-attribution of JMRI authors was not a cause of action under the US copyright law. So the difficult question is whether Katzer's act of deleting attribution information would lead to the infringement of the copyright of DecoderPro software. The trial court took the view that the attribution requirement was merely a contractual covenant, the breach of which is not a copyright infringement as such: 'The condition that the user insert a prominent notice of attribution does not limit the scope of the license' and it 'does not create liability for copyright infringement where it would not otherwise exist.'[97]

Failing to get an injunction from the trial court, Jacobsen then appealed the case to the US Court of Appeals for the Federal Circuit (CAFC), which reversed the trial court ruling by arguing that attribution of JMRI developers is a crucial *condition* for the public to use their copyrighted FOSS code in the first place. The failure to fulfil this condition would lead to infringement of copyright and thus give rise to the remedy of injunctive relief against the breacher. Most interestingly, the CAFC does not straightforwardly enforce JMRI authors' attribution for its own sake, but it unsurprisingly adopts the 'authorship-as-property' strategy through two steps. The first step denies that the case involves an 'attribution' dispute *per se*: 'Open source licensing restrictions are easily distinguished from mere "author attribution" cases. Copyright law does not automatically protect the rights of authors to credit for copyrighted materials.'[98] The second step dresses up the attribution requirement as a property claim[99] that furthers copyright owners' economic rights under the licensing conditions:

> The clear language of the Artistic License creates conditions to protect the *economic rights* at issue in the granting of a public license. These conditions govern the rights to modify and distribute the computer programs and files included in the downloadable software package. The attribution and modification transparency requirements directly serve to drive traffic to the open source incubation page and to inform downstream users of the project, which is a significant *economic* goal of the copyright holder that the law

---

96 535 F.3d 1373, 1376

97 The District Court's decision was quoted by CAFC, 535 F.3d 1373, 1380

98 535 F.3d 1373, FN5 1382

99 Note that the court's strategy to dress up an attribution claim as a property claim comes exactly from the 'legal lacuna' in US statutory law that refuses to give software programmers a paternity right. See above Section 2.2 of this article.

will enforce.[100] (emphasis added)

Although *Jacobsen*, as the first Anglo-American ruling that affirms the enforceability of a leading FOSS licence, is commendable in many ways[101], CAFC's overly economic interpretation of the case may also attract at least two types of criticism. The first criticises CAFC for making no real break from established proprietary information product licensing jurisprudence as started by law-and-economics judge Easterbrook over a decade before.[102] The second possible criticism is that the *Jacobsen* ruling is a missed opportunity to apply the craftsmanship theory in explaining FOSS programmers' legal persona. As the first criticism has been recently discussed in the legal literature[103], I will focus only on the second criticism, which is essentially a *normative* call from the craftsmanship model to recognise the FOSS programmers' pride of doing their job well for its own sake independent from economically motivated incentives.

4.2 Crafting Stewardship: A Normative Call for 'Authorship as Responsibility'

The craftsmanship model argues that attribution is not just a matter of authors' privately owned 'property', but is also deeply connected with their stewardship 'responsibility' to take care of the code in their craftsman persona. Again, in order to spell out this 'authorship-as-responsibility' argument, there are two significant issues worthy of attention in the context of the *Jacobsen* case. One concerns programmers' *motivation* (especially in terms of the reputational motivation that is linked with attribution) and the other concerns the *coordination* of individual contributions into a whole project (especially in terms of a project's collective reputation or goodwill). Both of these will be elaborated on in turn.

Firstly, the CAFC's reasoning seems to be underlined by an assumption that FOSS programmers are simply motivated by 'economic benefits' brought to them by contributing to a FOSS project. In particular, it points out that programmers' *reputational* gain can fall under these 'economic benefits': 'a programmer or company may increase its national or international reputation by incubating open source projects.'[104] This view is in tune with Eric Raymond's economic interpretation of FOSS programmers' motivation as driven by individual utility maximisation: The FOSS community is a bazaar-like market made up of 'a

---

100 535 F.3d 1373, 1382

101 Larry Rosen, 'Bad Facts Make Good Law: The Jacobsen Case and Open Source' (2009) 1 (1) *International Free and Open Source Software Law Review* <http://www.ifosslr.org/ifosslr/article/view/5/9> accessed 30 May 2013

102 *ProCD v Zeidenberg*, 86 F.3d 1447 (7th Cir.)

103 Benjamin I. Narodick, 'Smothered by Judicial Love: How *Jacobsen v. Katzer* Could Bring Open Source Software Development to a Standstill' (2010) 16 *Boston University Journal of Science and Technology Law* 264, 279-281; Chen Zhu, " 'Copyleft' Reconsidered: Why Software Licensing Jurisprudence Needs Insights from Relational Contract Theory" (2013) 22(3) *Social and Legal Studies* 289-308, 299-304

104 535 F.3d 1373, 1379

collection of selfish agents attempting to maximize utility which in the process produces a self-correcting spontaneous order'.[105] What is unique about this FOSS bazaar is that money is not primarily used as a measure of programmers' utility. Instead, FOSS programmers use 'reputational reward' as an alternative, which is believed to play a similar 'utility function' as money.[106] Under this logic, the more contribution is made by a programmer, the more reputational 'utility' can be generated to meet this creator's economic aspiration. It is interesting to note that this utilitarian interpretation of reputational reward is present not just in Raymond's writings, but is also accepted by some legal scholars who believe that highly skilled individuals including computer programmers can own their reputation as property. As Fisk comments: 'If professional reputation were property, it would be the most valuable property that most people own.'[107]

In contrast, the craftsmanship model is much more inclusive than CAFC's exclusively economic approach when dealing with motivation. It believes that FOSS programmers are motivated by a multiplicity of incentives and not just by the economic utility from the reputational reward. Most importantly, as already mentioned, FOSS programmer-craftsmen are primarily motivated by their intrinsic satisfaction from doing the work well for its own sake. This craftsmanship theory of motivation has been corroborated by an important empirical survey conducted by Lakhani and Wolf showing that the leading motivation of FOSS programmers is indeed what Torvalds calls the skill-based 'Entertainment with the capital E' comprising the intrinsic pleasure from programming and the prospect of improving programming skills.[108] Interestingly enough, the Raymondian reputational reward also exists according to the survey, but ranks relatively low in the list compared with the other motivational forces that are most commonly recognised by programmers themselves.[109]

Furthermore, even though the reputational reward is not the exclusive top motivation for FOSS programmers, it can still work *with* (rather than *against*) the craftsmanship model that focuses on the *quality* of programmers' codes. This is because FOSS projects with an effective attribution system can also use reputation as a *quality* measure of programmers' work. Reputation as a quality measure suggests that reputation is not just a programmer's private property, but that the programmer is under a responsibility for his reputation to be publicly assessed by peer programmers or software users. Weber finds that an individual

---

105 Note that Raymond's utilitarian theory unfortunately deviates from his own 'the-best-craftsmanship-wins' argument as mentioned before. Raymond, supra notes 19 & 74

106 Raymond, supra note 74

107 Fisk, supra note 22, 50

108 The survey finds that the following two motivational forces rank the highest: 'Code for project is intellectually stimulating to write' (44.9%) and 'Improve programming skills' (41.3%). Karim Lakhani and Robert Wolf, 'Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects' in Feller, Fitzgerald, Hissam & Lakhani (eds), *Perspective on Free and Open Source Software* (MIT Press, Cambridge, Mass. 2005) 13-14

109 The reputational motivation, according to the survey, is only at the bottom of the list. Ibid.

FOSS 'author is too close to the work and needs external measures of quality in order to know whether the work is good and how to improve it'.[110] In this sense, reputation is not just a proxy-measure for individual *utility* as assumed by Raymond, but is also a proxy-measure for the *quality* of code under the craftsmanship model:

> As is true of many technical and artistic disciplines, the quality of a programmer's mind and work is not easy for others to judge in standardized metrics. To know what is really good code and thus to assess the talent of a particular programmer takes a reasonable investment of time. The best programmers, then, have a clear incentive to reduce the energy that it takes for others to see and understand just how good they are. […] The programmer participates in an open source project as a demonstrative act to show the quality of her work. Reputation within a well-informed and self-critical community becomes the most efficient proxy measure for that quality.[111]

In addition to FOSS authors' motivation, the second issue of 'authorship as responsibility' is about the *responsibility of a core group of lead programmers to coordinate* individual contributions into a whole FOSS project, which again is not adequately dealt with under the 'authorship as property' model. When these lead programmers coordinate a certain project continuously over a long period, they should not only be credited for their individual contribution, but more significantly, also receive credit for their stewardship work that integrates other contributors' submissions into a collective whole. In other words, instead of owning reputation as property, lead programmers are also shouldering authorial responsibility for taking care of the whole project under concern. To illustrate, Torvalds (as the leader of the Linux kernel) may claim two types of authorship for his work. On the one hand, he is the *individual author* of the code written by him; on the other hand, he is also the *stewardship author* who reviews, approves and integrates other people's contribution into the mainline Linux kernel. The former is familiar to the Romantic mode of individuated authorship, while the latter is a less familiar one but is crucial to the success of a large-scale collaborative FOSS project.

It is worth noting that individual authorship and ownership may substantially overlap in a small budding project in its early formative stage, when a main programmer's individual contributions account for the greatest part of the program. At this stage, his significant individual authorship can easily give rise to project leadership, which is 'essentially the same as ownership' as observed by Weber.[112] However, when the project is scaled up into a much larger one, the lead programmer's individual authorship can be rapidly diluted to the extent that he can no longer justify his ownership/leadership of the whole program. If this programmer continues to be enthusiastic about taking the project forward, then the basis of his leadership must shift from an ever-dwindling *ownership* of the software to an

---

110 Weber, supra note 66, 141

111 Ibid., 142

112 Ibid., 166

ever-increasing *stewardship responsibility* in coordinating other programmers' contributions for the project.

This shift from ownership to stewardship is significant to a rounded understanding of project leaders' 'authorship as responsibility'. Countering the Romantic assumption of self-inspired authorship, the responsibility of FOSS leaders in stewardship seems to flesh out Kwall's thesis that conceptualises the 'author as steward'.[113] According to Kwall, there are two components in author-stewardship. The first comes from an awareness that an author himself is not the sole source of his creation. Instead, inspiration is externally endowed as a gift that enables the author to make his own creation.[114] In other words, the author-steward is not self-inspired, but receives external inspiration as a gift in which certain unearned value is bestowed upon him. In a large-scale FOSS project, it is clear that every programmer benefits from other people's contribution, and no one can claim to be the sole source of the whole program. Even many founding members of projects try hard to avoid reinventing the wheel if there are existing technologies available for reuse. Linus Torvalds, for example, did not start the Linux kernel from scratch in 1992, but obtained inspiration from the pedagogical Minix system initially developed by the Amsterdam-based computer scientist Andrew Tanenbaum in the late 1970s. Similarly, Stallman did not start the Emacs editor in the early 1980s from nothing. Instead, the program has been co-developed since the 1970s by a few programmers at the MIT Lab. The second component of author-stewardship goes against rewarding creators with exclusive ownership right. Instead it evokes a sense of responsibility to offer the author's work as a return gift back to the community from which the author gets his externally endowed inspiration in the first place. Or to put it in Kwall's words, it is the author's stewardship responsibility to participate in 'the cyclical dimension of creative enterprise'.[115] Hyde thinks that this responsibility actually comes from creators' 'labour of gratitude' which spurs creators to do something reciprocal for the external inspiration that is bestowed upon them early on.[116] In the history of FOSS development, Richard Stallman is a model of a programmer with a strong sense of stewardship responsibility to offer his software back to the community. When Stallman started his GNU project in 1983 (two years before the advent of the first copyleft licence in 1985), his initial announcement of the project clearly indicated that he was driven by an ethical responsibility to share his software with the community: 'I consider that the golden rule requires that if I like a program I must share it with other people who like it.'[117] His later experiment with the copyleft agreement, which makes programmers contribute modifications or improvements back to the community, further institutionalises programmers' stewardship responsibility through the mechanism of software licensing. Fusing the two components

---

113 Roberta Kwall, 'The Author as Steward "For Limited Times" ' (2008) *Boston University Law Review* 685-708

114 Ibid., 703

115 Ibid.

116 Lewis Hyde, *The Gift: Imagination and the Erotic Life of Property* (Vintage Books, New York 1983) 47

117 Stallman, *GNU Initial Announcement*, 1983 at <http://www.gnu.org/gnu/initial-announcement.html> accessed 30 May 2013

together, author-stewardship manages to bring to the foreground the 'responsibility' element in the 'authorship as responsibility' model, and it significantly departs from the 'authorship as property' model, which believes that the solitary self-inspired genius needs to be rewarded with private ownership for their creative works.

To summarise, the analysis of the *legal persona* of FOSS programmers above is quite different from that of their *authorial persona* in the previous section. The former is largely *normative* in the form of a proposal to build the craftsmanship element into a moral right regime independent from the property-oriented copyright, while the latter is predominantly *descriptive* in foregrounding the programmer-craftsmen's traits from the historical context of the hacker tradition. In other words, the craftsmanship theory does not only excel in describing FOSS programmers' authorial persona as craftspeople who are obsessed with the quality of their work, but it also has the potential in building a conceptual bridge towards programmers' legal persona as constructed by corresponding licensing schemes calling for a limited moral right regime of authorial attribution.

5. Conclusion

This article has sought to understand the authorial persona of FOSS programmers as shaped by their licensing schemes. It has argued that neither the Romantic-author vision nor the postmodern authorless creativity is suitable for defining FOSS programmers' authorial consciousness. Instead, it has found that Sennett's 'craftsmanship' theory—which explains craftsmen's intrinsic motive *to do a job well for its own sake*—is more adequate for addressing these programmers' authorial personas. The craftsmanship persona is also reflected in the prevalent 'attribution' clause in FOSS licensing, which enables the peer assessment of the quality of programmers' work associated with their reputation. However, current US copyright law does not statutorily recognise software programmers' moral right of attribution, but effectively blurs the distinction between authors' non-pecuniary moral rights and their economic rights as defined by the Berne Convention. It is proposed that FOSS authors' legal persona should depart from the copyright ownership and be re-anchored in their author-stewardship of the relevant projects, which are taken care of under FOSS programmer-craftsmen's authorial responsibility.